MICROCOMPUTER BASED MULTI-APPLIANCES CONTROL SYSTEM

BY

AYENI REUBEN E.
2001/12104EE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING SCHOOL OF ENGINEERING AND ENGINEERING TECHNOLOGY
FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA

A thesis submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering (B.Eng) degree in the Department of Electrical and Computer Engineering,
School of Engineering and Engineering Technology,
Federal University of Technology, Minna.

NOVEMBER, 2007

DECLARATION

I, Ayeni Reuben Erema declares that this project was my concept and was designed, construcied and tested under the supervision of Engr. Kolo J. G., Dept of Electrical and Computer Engineering.

[illegible]
Signature

03/12/09
Date

CERTIFICATION

I hereby certify that this project was carried out by **Mr. Ayeni Reuben Erema** of the Department of Electrical and Computer Engineering, School of Engineering and Engineering Technology, Federal University of Technology, Minna.

[illegible]
....
Engr. J.G. Kolo
(Supervisor)

03/12/07
....
Date

....
Engr. M.D. Abdullahi
(H.O.D)

....
Date

....
External Examiner

....
Date

DEDICATION

This project is dedicated to God- my destiny-maker, my way-maker, my hope giver and future builder, and my beloved parents, Mr. and Mrs Michael Babatunde Ayeni, who has always wanted the best for me in life.

ACKNOWLEDGEMENT

I thank the Sovereign God Jehovah for the gift of Life, sustenance and for being my inspiration throughout my study in the university.

I thank my supervisor, Engr J.G. Kolo for all his contributions and efforts towards the success of this project. To my H.O.D. Engr. M.D Abdullahi and every lecturer in Electrical and Computer Engineering Department, I say thank you for all your efforts to see that I am a success.

My profound gratitude goes to my mother, Mrs Ayeni Hellen Efurosibina, and Father, Mr Ayeni Michael Babatunde, for your sacrificial support all through my stay in school. My siblings, Oboro, Ededagobin, Mekabright and Mosifomokpo, for your loving support, I say thank you.

My dear aunty, Hajia Sikirat Saidu, I want to say thank you for your contribution. I also want to thank Otunba Ayeni Richard for all his support. All my friends, David Osama, Isaac and Richard Ogunmola, Comfort Jegede, Oluwatoyin Apata, Stephen Omotuke, Harry Gwar, Ronke Adesida, Titiolawunmi Madamidola, Rossy, I say thank you for the fun we share. To nurse Hannah Omokagbo, thank you for always been their.

To Nurse Obaro Blessing, "thank you" is just inadequate to appreciate you enough for all you have been to me all along. You are indeed a darling.

ABSTRACT

The Microcomputer Based Multi-Appliance Control System is built around an inverter driver IC, ULN 2803A. The circuit is designed to drive relay contacts, which then controls the loads. The circuit is pulsed by the parallel port of the PC. The output of the parallel port at any of the output pins connected to the input pins of the driver triggers the driver, which produces an inverted pulse of the input pulse at its corresponding output pins. These outputs energize the relays, thus controlling the loads. The driver is powered by a 12V output from the regulator circuit, while the Visual C program installed on the microprocessor of the Microcomputer controls the pulse generated at the parallel port.

TABLE OF CONTENT

CHAPTER ONE

GENERAL INTRODUCTION

1.1 INTRODUCTION

Aside the various usage the modern day computer offers, there seems to be an unending innovations and improvements on ways of making new and even better use of the computer. An outstanding usage of the computer is for communication, especially when it has to relate to peripheral devices like printers, mice, modems, scanners, digitizers, and many other devices that "talk to" and receive information from the microcomputer. Today, several industrial and scientific operations are controlled using the computer. With the control operation been able to be monitored through the monitor to know its state if it conforms to set standards, industrial and scientific activities has not only grown complex, but at the same time has increasingly been made relatively simpler. It is therefore imaginable, to consider the convenience this control feature of the computer has also allowed as it applies to its importance in security, household device management, e.t.c. This project therefore aims at the same and can be use to control the printer, loads & other household electrical appliances.

The hardware interface circuit comprises, voltage rectifier, voltage regulation, relay driver and relay sections. The microcomputer's parallel port was used to control these equipments. The software interface and program of controlling is resident at the computer's microprocessor and built around the Visual C++ language and compiled using Microsoft Visual Studio 6.

1.2 PROJECT OBJECTIVE

The aim of this project is to design and construct a peripheral hardware, interfaced with a microcomputer alongside support software that can be used for device control.

The following can therefore be achieved with this project:

i. Connected electrical appliances will be switched ON and OFF from the computer.

ii. The connected appliances can be regulated, for example the speed of Fan.

iii. These switching and regulating features can be pre- timed and events carried out automatically

iv. The processes above will be carried out with minimal human involvement.

v. The project switches four appliances, with one of the appliance configured for regulation.

1.3 METHODOLOGY

The microcomputer relates with peripheral devices through the interfaces called ports. The parallel port of the computer was used for interfacing the microcomputer with the external hardware system because of its advantage of been very fast in accessing data (byte-wise operation), which makes it the best suited port of the microcomputer in carrying out control activities.

1.4 PROJECT LAYOUT

This project thesis comprises of five chapters. The first chapter, chapter one **generally introduced** the project explaining the genesis of microcomputer (and by implication, the microprocessors), the tasks it can perform and its advantages. This chapter also states the objective of this project.

Chapter Two **"literature review"** deals with the theoretical and brief historical background of this project. Previous works as they relate to the project were also appraised.

Chapter Three **"design and implementation"** describes the stages of construction, the tools used to construct the hardware circuit, selection of choice components. The design and implementation of the software program is also covered in this chapter.

Chapter Four **"test, results and discussion"** describes the steps taken to test this project, alongside test and measurement methods used. The results obtained and the limitations of the project were also discussed.

Chapter Five **"Conclusion and Recommendation"** concludes with the great market potential of this wonderful project design. Under this same chapter, recommendation were made to individuals, industries, government and business agencies that the design can be used to control their numerous appliances from only one single computer system.

Finally, the **appendix** gives the source code listing of the software interface program used, full circuit diagram, and the graphical user interface.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

In the 1970s, National Aeronautics and Space Administration (NASA), an agency of the United States government, created the microchip- a tiny wafer of silicon that occupies a space smaller than a postage stamp. Computer components were placed on these microchips, hence computers required much less space than before. NASA produced these smaller computers in response to their need to send rocket ships to the moon with on-board computers. The computers on Earth could not provide split-second accuracy for rockets because radio waves took several seconds to travel between the Earth and the moon. Through development, these microchips became small enough so the computers could travel with a rocket and safely compute the rocket`s trajectory- *thus device control using the microcomputer was born!* This space program was not the only beneficiary of computer miniaturization. Because microchips became the heart of the microcomputer, computers could now fit on desktops [1]. These microcomputers cost much less than their larger counterparts, so many people started buying them. Thus, the home and small-business computer market was also born [2].

Computer miniaturization has also allowed for the development of other microprocessors which may not be computer based (that is, microcomputers with microprocessors that has no self in-built peripherals), but designed for other specific functions. However, these microprocessors are generally programmed using assembly language. This is not as flexible as using any of the high level language for the computer's microprocessor and it requires an understanding of the underlying architecture of the microprocessor to be programmed effectively [3]

Since its commercial availability more than twenty nine years ago, today, microprocessors are used throughout the electronic industry to perform tasks, ranging from the relatively simple- self control pumps, supermarkets cash registers, traffic light; through to complex applications such as autopilots in guided weapons, aircraft advanced communication systems, steer-able systems in drilling automation, e.t.c. The prospects in the use of microprocessors for device control is growing geometrically by every passing day- it thus seem that barely all human need for automation can be met: how challenging and interesting!

Nevertheless, for the computer and generally, microprocessors to carry out any function, owners still needed a way to program these machines. Over the years, several programming languages were designed to be "the only programming language one would ever need." PL/I was heralded as such in the early 1960s. It turned out to be so large and took so many system resources that it simply became another language programmers used, along with COBOL, FORTRAN, and many others. In the mid-1970s, Pascal was developed for smaller computers. Microcomputers had just been invented, and the Pascal language was small enough to fit in their limited memory space while still offering advantages over many other languages. Pascal became popular and is still used often today, but it never became *the* answer for all programming tasks, and it failed at being "the only programming language one would ever need." When the mass computer markets became familiar with C in the late 1970s, C also was promoted as "the only programming language one would ever need."

What has surprised so many skeptics is that C has practically fulfilled this promise! The appeal of C's efficiency, combined with its portability among computers, makes it the language of choice. Most of The UNIX operating system was written almost entirely in C. today's familiar spreadsheets, databases, and word processors are written in C. Now that C++ has improved on C, programmers have re-tooled their minds to think in C++ as well [4].

Bell Labs first developed the C programming language in the early 1970s, primarily so that Bell programmers could write their UNIX operating system for a new DEC (Digital Equipment Corporation) computer. Until that time, operating systems were written in assembly language, which is tedious, time-consuming, and difficult to maintain. The Bell Labs people knew they needed a higher-level programming language to implement their project quicker and create code that was easier to maintain. Because other high-level languages at the time (COBOL, FORTRAN, PL/I, and Algol) were too slow for an operating system's code, the Bell Labs programmers decided to write their own language. They based their new language on Algol and BCPL. BCPL strongly influenced C, although it did not offer the various data types that the makers of C required. After a few versions, these Bell programmers developed a language that met their goals well. C is efficient (it is sometimes called a high, low-level language due to its speed of execution), flexible, and contains the proper language elements that enable it to be maintained over time. In the 1980s, Bjourn Stroustrup, working for AT&T, took the C language to its next progression. Mr. Stroustrup added features to compensate for some of the pitfalls C allowed and changed the way programmers view programs by adding object-orientation to the language. The object-orientation aspect of programming started in other languages, such as Smalltalk. Mr.

Stroustrup realized that C++ programmers needed the flexibility and modularity offered by a true OOP programming language. C++ therefore became a powerful programming language designed for an advanced power—object-oriented programming (OOP). Now, Microsoft's Visual C++ is an improvement on C++, used to build a wide variety of applications with several in-built tools provided than is probably needed in any one individual application development effort. It has the particular advantage of writing less code words and much easiness in developing a graphic user interface (GUI). Visual C++ was used for the building of the software component of this project [5].

Summarily, the underlying power behind this growing electronic feature is the programs that drive these microprocessors. Depending on the nature of the control expected of the microprocessor, specific programs have to be written to control its operation. Such programs can be written in either the high level or low level program languages, depending on the type of processor used.

2.2 MICROPROCESSOR MODES OF DEVICE CONTROL

For the purpose of this project, the modes through which devices could be controlled using the microprocessor have been categorized into two, viz

i. **Microcomputer Based Control:** The microcomputer is a machine made up of several small microchips that uses its microprocessor with other peripheral to process data or information. The microcomputer therefore has a resident microprocessor. Therefore, when devices are controlled in this mode, the microprocessor of the computer is used for the control feature. Even though there are different means through which the microprocessor could be programmed, generally, a high level language is used (C++ the most common today). This has the added advantage of allowing the designer to incorporate a graphic user interface (GUI) The GUI, aside allowing the device to be easily accessed and controlled using the mouse or the keyboard, it also enables the user to know the state of the devices at any point in time when the program is initialized

ii. **Non-Microcomputer Based Control:** This control mode uses microprocessors which are not dependent on the operation of the microcomputer. Generally, a low level language is used, particularly the assembly language, to program it. To effectively program these processors, the basic underlining architecture of the microprocessor will have to be known.

2.2.1 Advantages of the Microcomputer Based Control:

1. Inexpensive, since there is no need to buy any microprocessor. Given a working microcomputer system, the program will only be written such that the necessary memory location on the microprocessor is affected.
2. It require less hardware interface circuitry
3. A high level language interface is utilized. Hence, no need to have a strong underlining knowledge of the microprocessor hardware architecture.
4. The microprocessor of the microcomputer is relatively consistent with its underlying hardware architecture. However, to program other microprocessors require strong knowledge of their different hardware architecture.

5. When an object- oriented programming language is used, it has an added advantage of one been able to develop a user friendly graphic user interface for the program, rather than running it in a usual DOS environment.

6. The program can easily be modified and improved on due to its flexibility, unlike the low level language which is tedious, time-consuming, and difficult to maintain.

2.3 MICROCOMPUTER INTERFACES

When controlling device from a microcomputer base, there is the need to build interfaces that will interact between the user and the computer (graphic user interface) and, the computer and the device (hardware interface). The design of these interfaces depends on the type of control to be implemented.

The Graphic User Interface (GUI) is a window displayed on the monitor when the control program is launched. It is the obvious aspect of the control program that allows the user to relate with the hardware interface From this window, the user determines the state of the device so wished to be controlled. It also indicates the status of the device if they are individually connected or not and give any other information of the system that will enhance its efficient and effective usage. The code program translates the user desire fed from the GUI to electrical impulses at the necessary port (in this case, the parallel port) of the computer system to be feed to the hardware interface.

The hardware interface is a built circuitry on a board that converts the impulse from the output data pins of the parallel port of the computer system to signals that finally determine the state of the devices to be controlled.

2.3.1 Microcomputer Ports

Generally, microcomputer ports are terminals of busses on its circuit board. The bus terminals herein referred to as ports are pins representing the different data, instruction, or control lines that make up the bus. In computers, ports are the main link between the microcomputer and other hardware devices external to it. They are used mainly for two functions: device control and communication. When used for communication, the computer communicates with the 'outside world' through different modes. These modes of communication and their connections enable transfer and sharing of files and documents, printing of documents, sending of electronic mail, access of software on other computers, and generally make two or more computers behave as a team. There are different types of ports. The categorization depends on their mode of receiving and giving out of data, and the speed at which these data are accessed per standard memory cycle. These modes of communication include.

ii. **Serial mode:** - This mode of operation makes use of the serial port for data transmission. This has basically two data lines: One for transmission and the other for receiving. To send a data in serial port, it has to be sent one bit after another with some extra bits like start bit, stop bit and parity bit to detect errors [6]

iii. **Universal serial bus (USB) mode:** - The universal serial bus is a relatively new bus technology compared to the serial and parallel ports. Essentially, the benefit of the USB specification is self-identifying peripherals (plug and play functionality), a feature that greatly ease installations.

iv. **Parallel mode:** - This mode of operation makes use of the parallel port for data transmission. Parallel port transmits data byte-wise. all the 8 bits of a byte will be sent to the port at a time and an indication will be sent in another line. It has some data lines, some control and some other handshaking lines [7]

Even though these ports all have the ability to be configured for control purpose, the parallel port is mainly used due to its speed, ruggedness, and the added pin features (the female D-type, resident on the motherboard has 25 pins for example)

CHAPTER THREE

DESIGN AND CONSTRUCTION

3.1 INTRODUCTION

Figure 3.1 shows the block diagram of the microcomputer based multi- appliance control system. From the diagram, it is clear that the whole operation of this system is based on the output signal generated from the parallel port of the computer system. The signals at the parallel port depend on the visual C++ program that is installed on the microprocessor of the computer, when initialized. The output pins of the parallel port directly handshake with the input pins of the buffer in the hardware interface circuit.

The basic electric components which consist of resistors, capacitors, diodes, relays, and integrated circuits (IC) were used in the design of the hardware interface of this project. This interface is built around the logic state of a buffer, ULN 2803A. This buffer gives an inverted logic state at its output pins, for every corresponding logic state at the input pins. These output logic state signals from the buffer are the triggering signals that control the state of the relays that in turn control the devices connected to them. This interface also has its own power circuit. This power circuit yields a regulated 12V output from a 15V step-down transformer. This output powers the buffer driver and the relay circuits

The microcomputer and the connected appliances are powered by the mains from PHCN at a maximum of 240V.

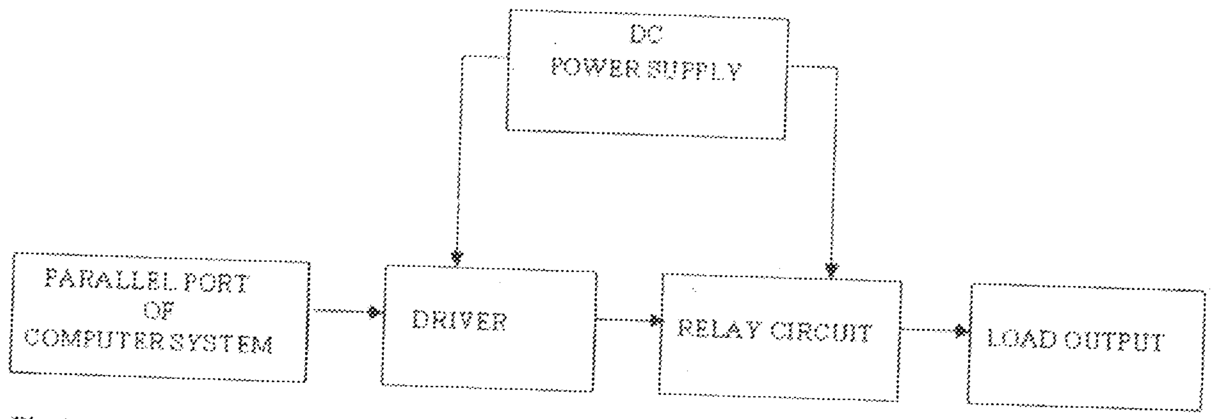


Fig 3.1 Microcomputer Based Multi- Appliance Control System`s Block Diagram

3.2 DESIGN SPECIFICATION

It is expected that the microcomputer based multi- appliance control system should meet the following specification:

i. The devices to be controlled should be powered by a 240 volts ac mains supply

ii. The hardware interface should be powered by a 240 volts ac mains supply, which is then converted to 12 volts by the use of voltage regulators after stepping it down by a transformer.

iii. In order to get an inverted logic state relative to the logic state of the parallel port, a buffer, ULN 2803A is used.

iv. The logic state output from the parallel port is determined by the operations carried out at the graphic interface level.

v. The input signals to the relays circuit should be fed from the output of the buffer.

3.3 THE MICROCOMPUTER

The microcomputer serves as the main control unit of the entire system. Input from the user is processed by it and compared with some preset conditions in the software programming. Depending on the condition that has been met by the input signals, the computer responds via the software control by sending out specific output.

The modern day computer is popularly called the PC after the IBM compatible computers, with barely the same underlying system architecture. Any of these PC can be programmed to perform any function. The various components, both internal and external, are interconnected by a series of electrical data highways over which data travels, as it complete the processing cycle that transform it from an input item to an output item. These "buses", connect these various components to the computers' central processing unit (microprocessor) and main memory (RAM).

Communication with each of these components is made possible by a unique address to each device by the microprocessor- an input- output port number. The PC has a built- in listing of all input- output units, each of which has their own port address.

In this project, the hardware interface is interfaced with the microcomputer via the parallel port. Only the data port of the parallel port was utilized.

3.4 COMPUTER PARALLEL PORT

A port contains a set of signal lines that the CPU uses to send or receive data with other components. They are used to communicate via modem, printer, keyboard, mouse etc. In signaling, open signals are "1" and close signals are "0" so it is like binary system. A parallel port sends 8 bits and receives 5 bits at a time. The parallel port or line printer terminal (LPT) port is a 25-pin D-type female connector available at the back of the computer. A basic IBM PC usually comes with one or two LPT ports. Figure 3.2 shows the diagram of a parallel port.

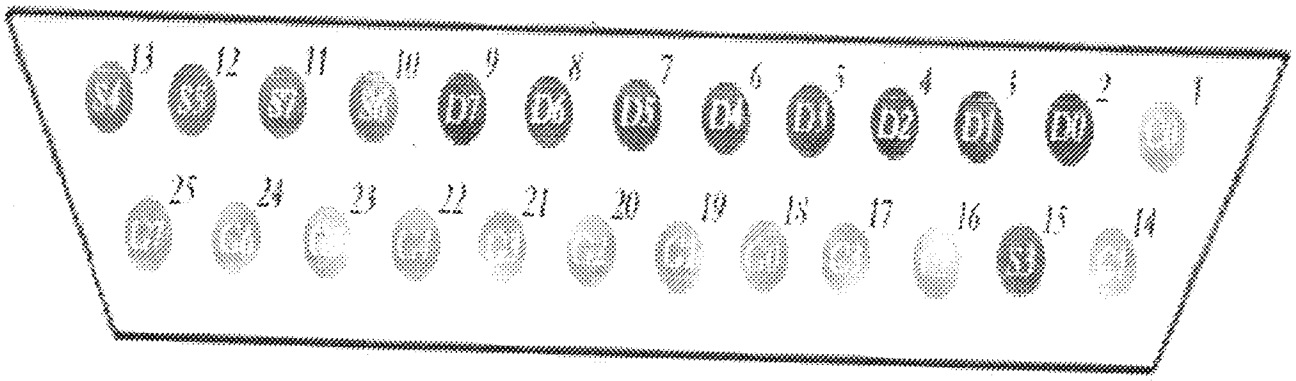


Fig. 3.2 Diagram of a parallel port showing its pins output

Normally, the parallel port is configured and used by the computer to communicate with the printer. One would then expect a conflict in task if a printer task and this project control task are initialized at the same time. The control pins of the parallel port help to take care of this problem, since the configuration of the control pins were not altered in any way while writing the code for this project.

3.4.1 Data Port

These are the pins 2 to 9, 8-bit port represented in figure 3.2 by D0 to D7. This port is purely a write-only port. This means it can be used only to output some data through it. It is these port pins that were programmed and used for the control feature of this project.

3.4.2 Status Port

These ports are made for reading signals. The range is like in data ports which are S0-S7. But S0, S1, S2 are invisible in the connector (as seen from figure 3.2, these pins are not shown). Though these pins are for reading signals, S0 is different. The S0 bit is for timeout flag in EPP (Enhanced Parallel Port) compatible ports. The address of this status port is 0x379. This often referred to as "DATA+1" and it can send 5 numeric data from the 10 - 11 - 12 - 13 - 15th pins. The functions of the Status pins are as highlighted below:

- S0. This bit becomes higher (1) if a timeout operation occurs in EPP mode
- S1: Not used (Maybe for decoration)
- S2: Mostly not used but sometime this bit shows the cut condition (PIRQ) of the port
- S3: If the printer determines an error it becomes lower (0). Which is called nError or nFault
- S4: It is high (1) when the data inputs are active. Which is called Select
- S5: It is high (1) when there is no paper in printer. Which is called PaperEnd, PaperEmpty or PError
- S6: It sends low impact signaling when the printer gets a one byte data. Which is called nAck or nAcknowledge
- S7. This is the only reversed pin on the connector (see my table in the article). If the printer is busy and it cannot get any additional data this pin becomes lower. Which is called Busy

3.4.3 Control Port

This port is usually used for outputting, but can also be used for inputting (hence, is capable of reading and writing). The range is like in data ports C0-C7 but C4, C5, C6, C7 are invisible in connector (hence not shown in figure 3.2). The functions of the control pins are as highlighted below:

- C0: This pin is reversed. It sends a command to read D0-D7 on the port. When the computer starts it is high in the connector. Which is called nStrobe
- C1: This pin is reversed. It sends a command to the printer to feed the next line. It is high in the connector after the machine starts. Which is called Auto LF

- C2: This pin is to reset the printer and clear the buffer. Which is called nInit, nInitialize
- C3: This pin is reversed. Sends a high (1) for opening data inputs. It is low after the machine starts. Which is called nSelectIn
- C4: Opens the cut operation for the printer. Not visible in the connector...
- C5: Sets the direction control in multidirectional ports. Not visible in the connector...
- C6: Not used and also Not visible in the connector...
- C7: Mostly not used but it is used as a C5 in some ports. Not visible in the connector...

3.4.4 Ground Pins

These are (G0 - G7) the pins from 18 to 25. These are mostly used for completing the circuit. Table 1 below shows the details of 25-pin parallel port

TABLE 1: Parallel Port Pin Details

Parallel port Pin no.	**Port signal Name**	**Direction**	**Hardware inverted**	**Register**
1	NStrobe	I/O	Yes	Control
2	D0	Out		Data
3	D1	Out		Data
4	D2	Out		Data
5	D3	Out		Data
6	D4	Out		Data
7	D5	Out		Data
8	D6	Out		Data

9	D7	Out		Data
10	NACK	In		Status
11	Busy	In	Yes	Status
12	Paper Out	In		Status
13	Select	In		Status
14	Nauto-Lf	I/O	Yes	Control
15	Nerror	In		Status
16	Ninitialize	I/O		Control
17	Nselect	I/O	Yes	Control
18-25	Ground	Gnd		

For the purpose of this project, the status and control ports were not used. The project is designed to use only the output port. Four of the output pins (D0 to D4) were configured to switch the target five appliances, while the remaining three pins (D5 to D7) were used to configure the control features of one of the appliance intended to be regulated.

The base address of the first parallel port (LPT1) is 0378 in hexadecimal (hex) notation (or 888 in decimal. The base address of the second parallel port (LPT2) is 0278 in hex. In this project, only LPT1 was used

3.5 SIGNAL GENERATOR

In any electronic project where switching is required, it is very important to generate a pulse signal that will be able to change between two voltage levels (i.e. for digital circuits).

so that one level will be for switching "off" (logic 0) and the other for switching "on" (logic 1), it could be the other way round depending on what is expected.

In this project, these pulses are generated through a computer program written in Visual C++ programming Language. This program determines the signals at the output pins of the computer parallel port.

3.5.1 The Visual C++ Development Environment

This is a window in the Microsoft Visual development environment. It has various areas with specific purpose. These arrears can be rearranged to customize the Developer Studio Environment to suit individuals particular development needs

i. The Workspace

This is the area on the left hand side of Developer Studio. It is the key to navigating the various pieces and parts of the development project. The workspace also allows one to view the parts of the application in three different ways:

- Class View allows one to navigate and manipulate the source code on a C++ class level.
- Resource View allows one to find and edit each of the various resources in the application, including dialog window designs, icons, and menus.
- File View allows one to view and navigate all the files that make up the application.

ii. The Output Pane

This might not be visible at the start of Visual C++. However, after the first compilation, it appears at the bottom of the Developer Studio environment and remains open until it is closed. The Output pane is where Developer Studio provides any information that it needs to give the programmer; where one sees all the compiler progress statements, warnings and error messages; and where the Visual C++ debugger displays all the variables with their current values as you step through the code. After the Output pane is closed, it reopens itself when Visual C++ has any message that it needs to display.

iii. The Editor Area

The area on the right side of the Developer Studio environment is the editor area. This is the area where all editing are performed when using Visual C++, where the code editor windows display when C++ source code is edited, and where the window painter displays when designing a dialog box. The editor area is also where the icon painter displays when the icons for use in the applications are been designed. The editor area is basically the entire Developer Studio area that is not otherwise occupied by panes, menus or toolbars.

iv. Menu Bars

At the first run of Visual C++, three toolbars display just below the menu bar. Many other toolbars are available in Visual C++, and they can be customized. New once can also be created. The three toolbars that are initially open are the following:

- The Standard toolbar contains most of the standard tools for opening and saving files, cutting, copying, pasting and a variety of other commands that one is likely to find useful.
- The Build minibar provides one with the build and run commands that the programmer most likely will use as he develops and test his applications. The full Build toolbar also lets one switch between multiple build configurations (such as between the Debug and Release build configurations).

3.5.2 Using The Application Wizard To Create The Application Shell

The AppWizard asks a series of questions about what type of application to be built and what features and functionality is needed. It uses this information to create a shell of an application that is immediately compiled and run. This shell provides the basic infrastructure that is needed to build the application around. The name entered at this stage for the dialog is what is seen given the project. For this project, 'ocean navigator' was used

3.5.3 Designing the Application Window (Interface)

Now that application shell is running, focus can then be put on the window layout of the application. Even though the main dialog window may already be available for painting in the editor area. This allows default texts to be deleted and replaced with costumed texts, command buttons, static text, edit box, check box, radio button, drop-down list box (also known as a combo box) and other basic Windows Controls, to be positioned, labeled, e.t.c

3.5.4 Adding Code to The Application

Codes are attached to the dialog through the Visual C++ Class Wizard. The Class Wizard can be used to build the table of Windows messages that the application might receive, including the functions they should be passed to for processing, that the Microsoft Foundation Classes (MFC) macros use for attaching functionality to window controls. To attach functionalities to applications the following steps are followed.

- To attach some functionality to buttons, right-click over the button and select Class Wizard from the pop-up menu. Once the button has been selected when you opened the Class Wizard, it is already selected in the list of available Object IDs.
- With the name of the button selected in the Object ID list, select BN_CLICKED in the list of messages and click Add Function. This opens the Add Member Function dialog. This dialog contains a suggestion for the function name. Click OK to create the function and add it to the message map

- After the function is added for the click message on the button, select the appropriate function in the list of available functions. Click the Edit Code button so that the cursor is positioned in the source code for the function, right at the position where you should add the functionality.

- Add the code in the Listing as appropriate, just below the TODO comment line.

- When the application is compiled and run, the button should display the appropriate message or carry out the required instruction

- Now that the application is functionally complete, finishing touches can be added like creating the dialog box icon, adding maximize and minimize buttons, e.t.c

The full code listing for this program is as shown in appendix [B]

3.5.5 Accessing The Output Port

The memory location corresponding to the data port is not bit addressable. This factor makes it difficult to address the data pins individually. Therefore, to address these pins, a byte of data must be sent to the base address-byte addressing (making the switching target of each device look impossible). To overcome this limitation the immediate base address has to be read and then ORed with a reference byte before sending to the base address. This reference byte should have all the bits low with only the bit corresponding with the pin of the target device's pin high. This action allows only the target device to be turned "ON". On the other hand, ANDing the base address with a reference byte results in an "OFF". Regardless of the state of the base address that is read, these operations stand true. Tables 2a and 2b highlight these operations.

TABLE 2a: Showing the events leading to accessing the output port for 'ON' operation.

STATE OF DATA IN MEMORY ADDRESS	PORT PIN LAYOUT								LOGIC EVENT	DEVICE STATE	DEVICE
	9	8	7	6	5	4	3	2			
INITIAL STATE	X	X	X	X	X	X	X	X			
REF. BITS	0	0	0	0	0	0	0	1			SECURITY LIGHT
FINAL STATE	X	X	X	X	X	X	X	1	OR	ON	
INITIAL STATE	X	X	X	X	X	X	X	X			
REF. BITS	0	0	0	0	0	0	1	0			TV
FINAL STATE	X	X	X	X	X	X	1	X	OR	ON	
INITIAL STATE	X	X	X	X	X	X	X	X			
REF. BITS	0	0	0	0	0	1	0	0			FRIDGE
FINAL STATE	X	X	X	X	X	1	X	X	OR	ON	
INITIAL STATE	X	X	X	X	X	X	X	X			
REF. BITS	0	0	0	0	1	0	0	0			FAN SPEED 1
FINAL STATE	X	X	X	X	1	X	X	X	OR	ON	
INITIAL STATE	X	X	X	X	X	X	X	X			
REF. BITS	0	0	0	1	1	0	0	0			FAN SPEED 2
FINAL STATE	X	X	X	1	1	X	X	X	OR	ON	
INITIAL STATE	X	X	X	X	X	X	X	X			
REF. BITS	0	0	1	1	1	0	0	0			FAN SPEED 3
FINAL STATE	X	X	1	1	1	X	X	X			

TABLE 2b: Showing the events leading to accessing the output port for 'OFF' operation.

STATE OF DATA IN MEMORY ADDRESS	PORT PIN LAYOUT								LOGIC EVENT	DEVICE STATE	DEVICE
	9	8	7	6	5	4	3	2			
INITIAL STATE	X	X	X	X	X	X	X	X			SECURITY LIGHT
REF. BITS	1	1	1	1	1	1	1	0			
FINAL STATE	X	X	X	X	X	X	X	0	AND	OFF	
INITIAL STATE	X	X	X	X	X	X	X	X			TV
REF. BITS	1	1	1	1	1	1	0	1			
FINAL STATE	X	X	X	X	X	X	0	X	AND	OFF	
INITIAL STATE	X	X	X	X	X	X	X	X			FRIDGE
REF. BITS	1	1	1	1	1	0	1	1			
FINAL STATE	X	X	X	X	X	0	X	X	AND	OFF	
INITIAL STATE	X	X	X	X	X	X	X	X			FAN SPEED 1
REF. BITS	1	1	1	1	0	1	1	1			
FINAL STATE	X	X	X	X	0	X	X	X	AND	OFF	
INITIAL STATE	X	X	X	X	X	X	X	X			FAN SPEED 2
REF. BITS	1	1	1	0	0	1	1	1			
FINAL STATE	X	X	X	0	0	X	X	X	AND	OFF	
INITIAL STATE	X	X	X	X	X	X	X	X			FAN SPEED 3
REF. BITS	1	1	0	0	0	1	1	1			
FINAL STATE	X	X	0	0	0	X	X	X	AND	OFF	

The GUI window displayed has time display, auto time control setting (time start and time stop), ON and OFF button, e.t.c., for each device. The ON and OFF button for each device is tied to a corresponding data pin of the parallel port. At any time the button is clicked ON, it places a digital "1" at the data pin of the parallel port tied to it. On the other hand, if OFF, a digital "0" is placed at the same pin. Since these data lines are fed directly to the input of the driver, the instantaneous digital value at the parallel port is fed to it. The function of the driver is to invert this state at the corresponding output pin. Hence, a logical '1' at the input of the driver implies a logical '0' at its corresponding output pin. This logical '0' at its output is needed to turn ON the relays.

The auto time control setting is configured to work with the microcomputer time configuration on 24 hourly bases. So, the program compare the set time with the system time from the time the time control is triggered. Once the set time equals that of the system for the 'time start', the program automatically trigger 'ON' the corresponding device. On the other hand, once the "time stop" is reached, the program automatically triggers 'OFF' the corresponding device. The two timers can be set to run simultaneously.

3.6 POWER SUPPLY

The power supply functions basically to provide the necessary dc voltage with low level of ac ripple and with good stability and regulation. It is important to state here that sometimes, the source of this power supply could be a battery. However, often, power is obtained from a unit that converts the normal single- phase ac mains supply from local source (e.g. PHCN) 240 volts to some different value of ac voltage suitable for the circuitry in concern. One of the various methods of achieving a stable dc voltage from ac mains is by the use of linear stabilizer.

In the case of this project, a 240/15 volts centre tapped step-down transformer was used to step down the ac voltage. The desired 12 volts for the logic circuit respectively was then realized using a full-wave bridge rectifier to a steady dc voltage with a choice voltage regulator IC, LM 7812, connected in parallel to the output of the transformer.

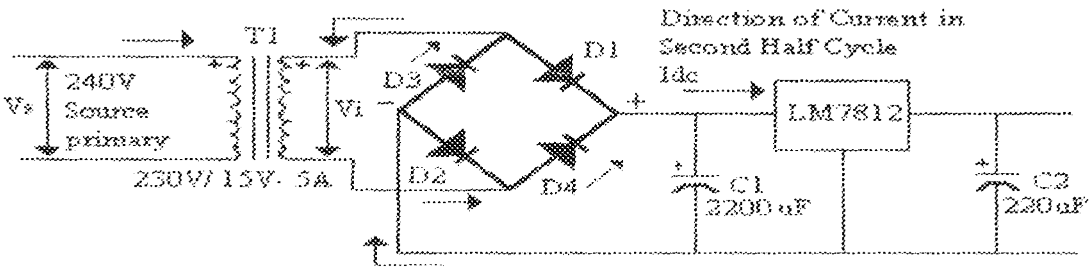


Fig 3.3: The power supply circuitry

The steady dc voltage realized needed to be smoothened to remove the ripple effect caused by the ac components. This smoothening action is performed by a large value electrolytic capacitor connected across the DC supply to act as a reservoir, supplying current to the output when the varying DC voltage from the rectifier is falling. This action significantly increases the average DC voltage to almost the peak value (1.4 × RMS value). This smoothing is not perfect due to the capacitor voltage falling little as it discharges, giving a small ripple voltage. For many circuits a ripple which is 10% of the supply voltage is satisfactory with a larger capacitor giving fewer ripples. The capacitor value must be doubled when smoothing half-wave DC. From the power supply circuitry diagram, C1 and C2 is use or smoothening.

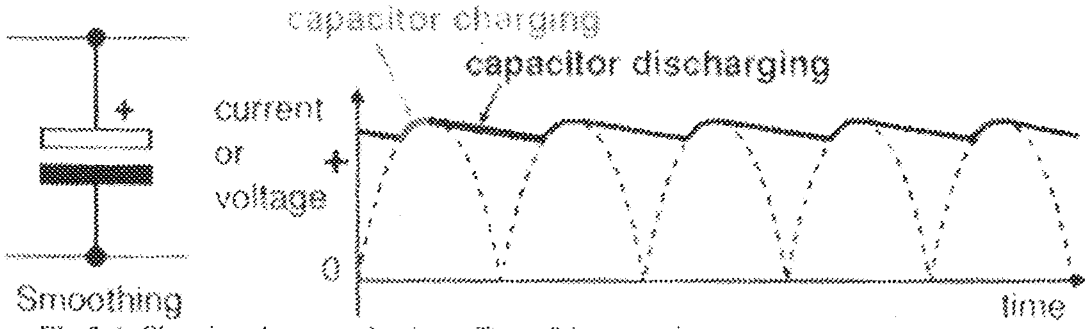


Fig 3.4: Showing the smoothening effect of the capacitor

3.7 THE RELAY

A relay is a coil with a specific inductance (LC, in Henry) that causes a contact to open or close when a specified current Ion, in Amperes, charges it. It is therefore a type of switch whose operation is based on electromagnetic principles. In other words it is activated when a current is applied to it. The magnetic relay can be used as a normally open or normally closed relay. It is activated when the current in the energizing circuit exceeds the value of the switch-on current, I_{on}. During operation, the contact switches from the normally closed terminals to the normally opened terminals. The relay will remain on as long as the current in the circuit is greater than holding current, I_{hd}. The value of I_{hd} must be less than that of I_{on}.

The contact remains in the same position until the current falls below the holding value, I_{hd} in Amperes, at which point it returns to its original position.

The energizing coil of the relay is modeled as an inductor, and the relay's switching contact is modeled as a resistors. The circuit diagram of a relay is shown figure

This unit directly interacts with the connected appliances and is responsible for switching the appliance on and off, alongside the regulating operation. The circuitry is given below.

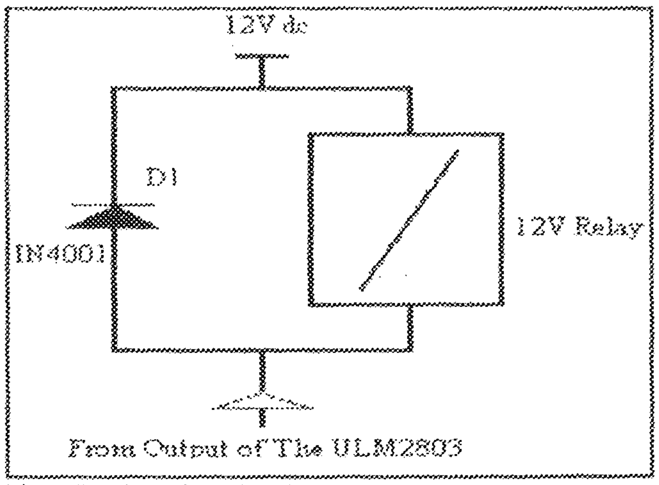


Fig 3.5: The relay circuitry

From the circuit above, the diode functions as a free wheeling diode. Thus, it prevents the switching device (Driver IC) from the charge reversal of the inductor (Relay).

3.8 THE DRIVER

The ULN 2803A is one of the series of the ULN28xxA, high-voltage, high-current Darlington arrays that are ideally suited for interfacing between low-level logic circuitry and multiple peripheral power loads. Typical loads include relays, solenoids, stepping motors, magnetic print hammers, multiplexed LED and incandescent displays, and heaters. All devices feature open-collector outputs with integral clamp diodes. The ULN2803A is furnished in 18-pin dual in-line plastic packages with series input resistors selected for operation directly with 5 V TTL or CMOS. It can handle numerous interface needs — particularly those beyond the capabilities of standard logic buffers. The ULN2803A is one of the standard Darlington arrays. The output is capable of sinking 500 mA and will withstand at least 50 V in the OFF state. Outputs may be paralleled for higher load current capability. The ULN2803A device is rated for operation over the temperature range of -20°C to +85°C.

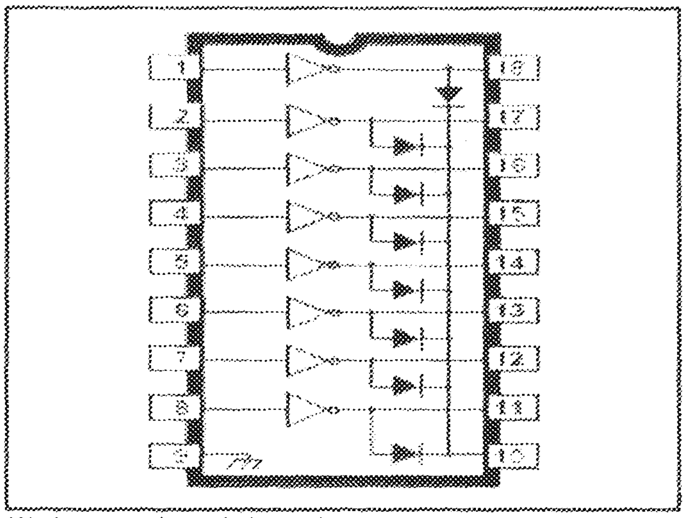


Fig3.6: Pin Out of ULM2803

This unit is used to drive the relay, this is necessary because the output of the computer cannot drive the relay directly. It is also important because it also isolate the parallel port from electromagnetic switch.

The full circuit diagram is shown in appendix [A]

CHAPTER FOUR

TEST, RESULT AND DISCUSSION

4.0 PRODUCT CHARACTERISTIC FOCUS

In designing and constructing the Microcomputer multi appliance control system, the following factors were carefully put into consideration. They include:

- Reliability
- Durability
- Efficiency
- Affordability
- Effectiveness
- Low purchase cost

4.1 HARDWARE CIRCUIT CONSTRUCTION

The construction and assembling of the hardware components was divided into three stages, which include:

i. **Mounting of components on Breadboard:** - This stage was the experiment stage. It involved placing the components on the Breadboard and using jumpers to link the major components as appropriate. The multi-meter was used to test the signals expected at certain nodes. The circuit was then adjusted to meet the target specification by carefully adjusting the values of the critical components. After all the components have been mounted and the circuit was tested working and meeting the targeted specification, it was then transferred to the next stage- Vero board

ii. **Mounting of components on Vero board:** - A suitable Vero board was cut into a suitable size that should contain all components involved. The power circuit was first constructed, followed by other leading component. The soldering operation was smoothly and neatly done, carefully avoiding short circuit and excessive heating up of the components Again, the circuit was tested for errors. With the circuit operating very close to the specification, the circuit was housed- which is the next stage.

iii. **Packaging:** - The circuit, having been found working operationally, was housed in a transparent plastic case as a complete unit. The choice of a transparent plastic housing was made due to its availability, convenience, low cost, and attractiveness.

4.2 SOFTWARE DEVELOPMENT

The development of the software involves the following three stages:

i. **Building of the graphic user interface (GUI):** - The GUI was built using the powerful Microsoft Visual C++ development environment. Using the in- built, dialog-based MFC package, the design of the GUI was implemented. After the dialog was invoked, it was first 'built' and 'run', and then 'executed', to allow the program generate the '.exe' file. The various controls were then put on the dialog and sized accordingly. Furthermore, identity names were attached to the various controls alongside variable names for controls that require such. Other attractive touches were given the interface to make it more user-friendly. The interface was 'built' and 'run'. With no error message, the codes were then added.

ii. **Adding the program code:** - The program codes were attached to each of the controls by calling their identity names from the code editor. After each control has its codes attached to it, it was 'built' and 'run' to check it for errors. The process was repeated for all the controls until certain that there were no errors. The general program was then 'built' and 'run'. With no error message recorded, the software package was ready to carry out its control of the hardware interface.

4.2.1 Port Driver

Due to security reasons Microsoft operating systems do not allow any how access to the ports of microcomputers. To access the ports, especially the parallel port, specially built software drivers, called Direct Link Libraries (DLL) are required to allow the port to be configured for use. The drivers needed to access the parallel port are the INPOUT.DLL and INPUT.DLL. These drivers will need to be installed on the system before the software program can access the computer`s parallel port.

4.3 TESTING

At this stage, the printer cable was connected from the microcomputer to the hardware interface and the various loads connected to their respective sockets on the hardware interface. The loads and the microcomputer were powered from the mains. The *ocean navigator* was initialized from the microcomputer. From the interface, the 'ON' and 'OFF' buttons corresponding to the various loads was clicked. With the loads behavior corresponding to the desired specification, it confirms that the entire system was functioning appropriately. The speed control of the Fan was also tested. With the obvious varying speed of the fan in response to the clicking event of the buttons corresponding to the different speed, it also confirmed that the speed control was operating fine. Again, the timer event was tested. Various 'START' and 'STOP' time were entered at the interface for the different loads. Since the loads came 'ON' and 'OFF' respectively, it then also confirms that the system was operating according to the desired specification

4.4 DISCUSSION OF RESULT

From the test carried out, it was seen that each of the loads came 'ON' when the ON-button was clicked, and 'OFF' when the same button was clicked again. These same events occur when the timer function is invoked. When the set 'START' time of the timer reach its equivalent system time, say 00:00:00 hrs, it triggers the ON-button. On the other hand, when the set 'STOP' time of the timer reach its equivalent system time, say 00.00:00 hrs, it triggers the button OFF.

Clicking the shutdown button saves the settings of *ocean navigator* and any other program running on the system and shutdown the system. Enabling the shutdown timer and setting the timer at a shutdown time triggers the shutdown event, when the set time reaches the equivalent system time.

The *ocean navigator*, when initialized also runs its own time from the timer display unit. The time displayed was seen to be the same with that displayed from the system.

Finally, when the close button is clicked, the program is brought to an end, with the setting saved, and the loads retaining these saved settings.

The result is however based on the assumption that the initial condition of the loads is such that the main power supply is not cut off, such as when power failure from *PHCN* occurs. If this situation thus occurs and the system settings remain constant, the state of the loads will remain unchanged when there is an eventual restoration of power.

Lastly if for any reason the system unit need be put-off, and every other factors remain constant, the state of the load will retain their states till further changes are made to them when the system is powered again.

4.5 PRECAUTIONS

During the construction of this project, the following precautions were taken:

i. The various components were prevented from heat related damage by making sure that the solder was carefully applied

ii. Little but enough solder was applied to any joint to ensure proper contact of the components

iii. Care was taken to ensure proper soldering of each joint, so that the lead of individual joint would not heat away.

iv. Heat sink was used to conduct heat away

v. It was also ensured that the soldering- iron temperature was not too high to avoid damage resulting from over-heating

vi. Sensitive parts of the circuit were properly insulated to prevent short circuit.

vii. The circuit was firmly screwed to the casing for rigidity to prevent unexpected removal of wires.

viii. Extreme care was taken to ensure that there were no conflicts with the use identity names when entering the codes for the various controls.

ix. Also the syntax for entering the codes was carefully followed to avoid errors

x. I also made sure that I was careful enough not to delete any of the in-built, resident code that enhances the operation of the program.

4.6 DIFFICULTIES ENCOUNTERED

Basically, the difficulties encountered were from the development of the software interface. It was difficult avoiding any conflict of controls identity names or even spelling them wrongly. It was also a tough time combating with viruses.

CHAPTER FIVE

5.1 CONCLUSION

The design of this project was quite very simple and straight forward. The components used were readily available and affordable in the market. The objectives as earlier stated were also achieved satisfactorily

5.2 RECOMMENDATION

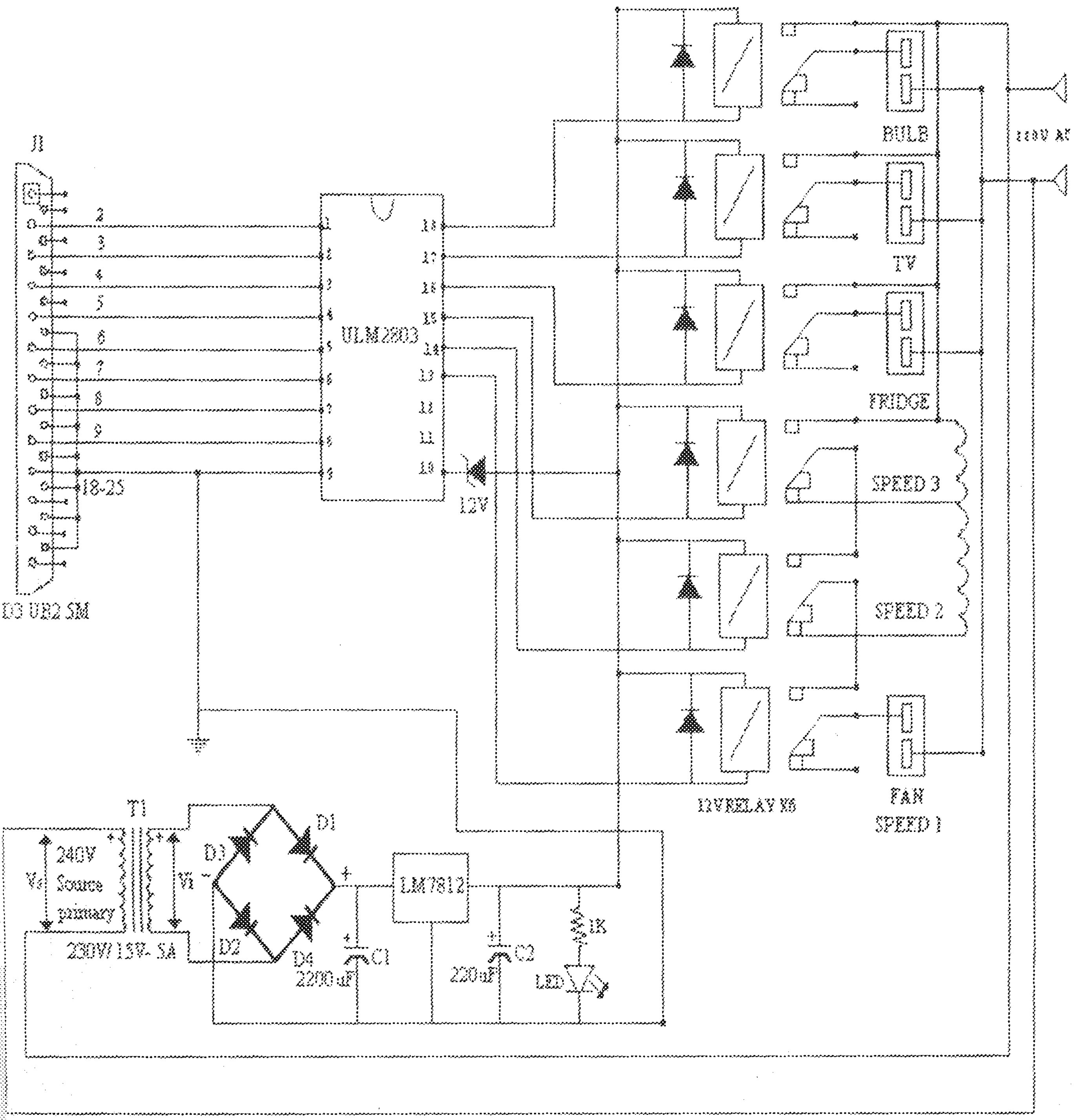
I will recommend that the parallel port of the microcomputer be put to better use because of its obvious advantages. All eight output data lines can be put to use, thereby enhancing greater control and capabilities. More devices like heat or smoke detector, temperature sensor, e.t.c. can be interfaced with it Motor control could also be incorporated into the design.

When more devices are needed to be interfaced than the eight supported basically by the parallel port, cascading can be employed using decoders. Depending on the need of the designer, many devices (unending number) can be interfaced using this means.

Finally, individuals, business and corporate organizations should take the advantage this project offers to improve on the control of their electrical appliances with less human interference. The same applies to the industrial organizations. It will be fun to have their heavy machines and equipments switched ON and OFF by "themselves" and their operations even regulated. Feedback path could also be incorporated to allow the user observe the state of the control work been carried out at any particular time. This project gives the necessary background information for such improvements.

APPENDIX A

CIRCUIT DIAGRAM



APPENDIX B

PROGRAM SOURCE CODE

```
// DeXDlg.cpp : implementation file
//

#include "stdafx.h"
#include "DeX.h"
#include "DeXDlg.h"
#include "conio.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define DATA 0x378
#define STATUS 0x379
#define CONTROL 0x37A


short _stdcall Inp32(short portaddr);
void _stdcall Out32(short portaddr, short datum);
/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
```

```
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDeXDlg dialog

CDeXDlg::CDeXDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CDeXDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDeXDlg)
	m_BBOnOff = _T("Off");
	m_GTOnOff = _T("Off");
	m_SSOnOff = _T("Off");
	m_SWOnOff = _T("Off");
	m_sTime = _T("");
	m_BBApplianceStatus = _T("Not working...");
	m_GTApplianceStatus = _T("Not working...");
	m_SSApplianceStatus = _T("Not working...");
	m_SWApplianceStatus = _T("Not working...");
	m_SWEnableTiming = TRUE;
	m_SSEnableTiming = TRUE;
	m_GTEnableTiming = TRUE;
	m_BBEnableTiming = TRUE;
	m_BBPortStatus = _T("Not working...");
```

```
        m_GTPortStatus = _T("Not working...");
        m_SSPortStatus = _T("Not working...");
        m_SWPortStatus = _T("Not working...");
        m_SWStartedTime = _T("00:00");
        m_SSStartedTime = _T("00:00");
        m_GTStartedTime = _T("00:00");
        m_BBStartedTime = _T("00:00");
        m_SWStartTime = _T("00:00");
        m_SSStartTime = _T("00:00");
        m_GTStartTime = _T("00:00");
        m_BBStartTime = _T("00:00");
        m_BBStopTime = _T("00:00");
        m_GTStopTime = _T("00:00");
        m_SSStopTime = _T("00:00");
        m_SWStopTime = _T("00:00");
        m_pin2 = FALSE;
        m_pin3 = FALSE;
        m_pin4 = FALSE;
        m_pin5 = FALSE;
        m_BBEnableDisable = _T("Timing\nDisable");
        m_GTEnableDisable = _T("Timing\nDisable");
        m_SSEnableDisable = _T("Timing\nDisable");
        m_SWEnableDisable = _T("Timing\nDisable");
        m_ShutDownTime = _T("00:00");
        m_ShutDown = FALSE;
        m_EnableShutdown = FALSE;
        m_ESD = _T("");
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CDeXDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDeXDlg)
        DDX_Text(pDX, IDC_STATICBB, m_BBOnOff);
        DDX_Text(pDX, IDC_STATICGT, m_GTOnOff);
        DDX_Text(pDX, IDC_STATICSS, m_SSOnOff);
        DDX_Text(pDX, IDC_STATICSW, m_SWOnOff);
        DDX_Text(pDX, IDC_STATICTIME, m_sTime);
        DDX_Text(pDX, IDC_BBAPST_EDIT, m_BBApplianceStatus);
        DDX_Text(pDX, IDC_GTAPST_EDIT, m_GTApplianceStatus);
        DDX_Text(pDX, IDC_SSAPST_EDIT, m_SSApplianceStatus);
        DDX_Text(pDX, IDC_SWAPST_EDIT, m_SWApplianceStatus);
        DDX_Check(pDX, IDC_SWEnableTiming, m_SWEnableTiming);
        DDX_Check(pDX, IDC_SSEnableTiming, m_SSEnableTiming);
        DDX_Check(pDX, IDC_GTEnableTiming, m_GTEnableTiming);
        DDX_Check(pDX, IDC_BBEnableTiming, m_BBEnableTiming);
        DDX_Text(pDX, IDC_BBPTST_EDIT, m_BBPortStatus);
        DDX_Text(pDX, IDC_GTPTST_EDIT, m_GTPortStatus);
        DDX_Text(pDX, IDC_SSPTST_EDIT, m_SSPortStatus);
        DDX_Text(pDX, IDC_SWPTST_EDIT, m_SWPortStatus);
        DDX_Text(pDX, IDC_SWSDTM_EDIT, m_SWStartedTime);
        DDV_MaxChars(pDX, m_SWStartedTime, 8);
        DDX_Text(pDX, IDC_SSSDTM_EDIT, m_SSStartedTime);
        DDV_MaxChars(pDX, m_SSStartedTime, 8);
        DDX_Text(pDX, IDC_GTSDTM_EDIT, m_GTStartedTime);
        DDV_MaxChars(pDX, m_GTStartedTime, 8);
        DDX_Text(pDX, IDC_BBSDTM_EDIT, m_BBStartedTime);
        DDV_MaxChars(pDX, m_BBStartedTime, 8);
        DDX_Text(pDX, IDC_SWSTTM_EDIT, m_SWStartTime);
        DDV_MaxChars(pDX, m_SWStartTime, 8);
        DDX_Text(pDX, IDC_SSSTTM_EDIT, m_SSStartTime);
        DDV_MaxChars(pDX, m_SSStartTime, 8);
        DDX_Text(pDX, IDC_GTSTTM_EDIT, m_GTStartTime);
        DDV_MaxChars(pDX, m_GTStartTime, 8);
        DDX_Text(pDX, IDC_BBSTTM_EDIT, m_BBStartTime);
        DDV_MaxChars(pDX, m_BBStartTime, 8);
        DDX_Text(pDX, IDC_BBSPTM_EDIT, m_BBStopTime);
        DDV_MaxChars(pDX, m_BBStopTime, 8);
        DDX_Text(pDX, IDC_GTSPTM_EDIT, m_GTStopTime);
        DDV_MaxChars(pDX, m_GTStopTime, 8);
        DDX_Text(pDX, IDC_SSSPTM_EDIT, m_SSStopTime);
        DDV_MaxChars(pDX, m_SSStopTime, 8);
        DDX_Text(pDX, IDC_SWSPTM_EDIT, m_SWStopTime);
        DDV_MaxChars(pDX, m_SWStopTime, 8);
        DDX_Check(pDX, IDC_Pin2, m_pin2);
        DDX_Check(pDX, IDC_Pin3, m_pin3);
        DDX_Check(pDX, IDC_Pin4, m_pin4);
        DDX_Check(pDX, IDC_Pin5, m_pin5);
        DDX_Text(pDX, IDC_BBEnableDisable, m_BBEnableDisable);
        DDX_Text(pDX, IDC_GTEnableDisable, m_GTEnableDisable);
        DDX_Text(pDX, IDC_SSEnableDisable, m_SSEnableDisable);
        DDX_Text(pDX, IDC_SWEnableDisable, m_SWEnableDisable);
        DDX_Text(pDX, IDC_SHUTDOWNTIME, m_ShutDownTime);
        DDX_Check(pDX, IDC_ENABLESHUTDOWN, m_EnableShutdown);
        DDX_Text(pDX, IDC_ESD, m_ESD);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDeXDlg, CDialog)
```

```
	//{{AFX_MSG_MAP(CDeXDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_EXIT, OnExit)
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_BBEnableTiming, OnBBEnableTiming)
	ON_BN_CLICKED(IDC_GTEnableTiming, OnGTEnableTiming)
	ON_BN_CLICKED(IDC_SSEnableTiming, OnSSEnableTiming)
	ON_BN_CLICKED(IDC_SWEnableTiming, OnSWEnableTiming)
	ON_BN_CLICKED(IDC_Pin2, OnPin2)
	ON_BN_CLICKED(IDC_Pin3, OnPin3)
	ON_BN_CLICKED(IDC_Pin4, OnPin4)
	ON_BN_CLICKED(IDC_Pin5, OnPin5)
	ON_BN_CLICKED(IDC_ENABLESHUTDOWN, OnEnableshutdown)
	ON_WM_HELPINFO()
	ON_BN_CLICKED(IDC_Pin6, OnPin6)
	ON_BN_CLICKED(IDC_Pin7, OnPin7)
	ON_BN_CLICKED(IDC_Pin8, OnPin8)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDeXDlg message handlers
long int ctr=0;
#include <iomanip.h>
#include <fstream.h>

BOOL CDeXDlg::OnInitDialog()
{


  Out32(DATA,255);

	Out32(CONTROL,0);


    LoadSettings();
    SetTimer(ID_CLOCK_TIMER, 1000, NULL);


	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
```

```
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
                }
        }

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                 // Set big icon
        SetIcon(m_hIcon, FALSE);                // Set small icon

        // TODO: Add extra initialization here

        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CDeXDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CDeXDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CDeXDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CDeXDlg::OnExit()
{
        // TODO: Add your control notification handler code here
  m_pin9=0;
        UpdateData(FALSE);
        ChangePin();
        OnOK();
}



void CDeXDlg::OnTimer(UINT nIDEvent)
{

  StatusPins();
        // TODO: Add your message handler code here and/or call default
        // Get the current time
        ctr++;
                if(ctr==1)
                {
     CTime curTime = CTime::GetCurrentTime();

     // Display the current time
     m_sTime.Format("%d:%d", curTime.GetHour(),
        curTime.GetMinute(),
                curTime.GetSecond());

     // Update the dialog
     UpdateData(FALSE);

     TimmerEvents();
                }
        else
        {
                if(ctr%60==0)
        {
     CTime curTime = CTime::GetCurrentTime();

     // Display the current time
     m_sTime.Format("%d:%d", curTime.GetHour(),
        curTime.GetMinute(),
                curTime.GetSecond());

     // Update the dialog
     UpdateData(FALSE);

     TimmerEvents();
                }

        }
        CDialog::OnTimer(nIDEvent);

}

void CDeXDlg::OnBBEnableTiming()
{
        // TODO: Add your control notification handler code here

        EnableTiming();
}

void CDeXDlg::OnGTEnableTiming()
{
        // TODO: Add your control notification handler code here
        EnableTiming();
}

void CDeXDlg::OnSSEnableTiming()
{
        // TODO: Add your control notification handler code here
        EnableTiming();
}

void CDeXDlg::OnSWEnableTiming()
{
        // TODO: Add your control notification handler code here
        EnableTiming();
}

void CDeXDlg::EnableTiming()
{
 UpdateData(TRUE);

        if(m_BBEnableTiming==TRUE)
                {m_BBEnableDisable="Timing\nEnabled";
      GetDlgItem(IDC_BBSTTM_EDIT)->EnableWindow(FALSE);
      GetDlgItem(IDC_BBSPTM_EDIT)->EnableWindow(FALSE);}
        else{m_BBEnableDisable="Timing\nDisabled";
                GetDlgItem(IDC_BBSTTM_EDIT)->EnableWindow(TRUE);
      GetDlgItem(IDC_BBSPTM_EDIT)->EnableWindow(TRUE);}

        if(m_GTEnableTiming==TRUE)
                {m_GTEnableDisable="Timing\nEnabled";
      GetDlgItem(IDC_GTSTTM_EDIT)->EnableWindow(FALSE);
      GetDlgItem(IDC_GTSPTM_EDIT)->EnableWindow(FALSE);}
        else{m_GTEnableDisable="Timing\nDisabled";
            GetDlgItem(IDC_GTSTTM_EDIT)->EnableWindow(TRUE);
      GetDlgItem(IDC_GTSPTM_EDIT)->EnableWindow(TRUE);}

        if(m_SSEnableTiming==TRUE)
                {m_SSEnableDisable="Timing\nEnabled";
                GetDlgItem(IDC_SSSTTM_EDIT)->EnableWindow(FALSE);
      GetDlgItem(IDC_SSSPTM_EDIT)->EnableWindow(FALSE);}
        else{m_SSEnableDisable="Timing\nDisabled";
           GetDlgItem(IDC_SSSTTM_EDIT)->EnableWindow(TRUE);
      GetDlgItem(IDC_SSSPTM_EDIT)->EnableWindow(TRUE);}

    if(m_SWEnableTiming==TRUE)
                {m_SWEnableDisable="Timing\nEnabled";
        GetDlgItem(IDC_SWSTTM_EDIT)->EnableWindow(FALSE);
    GetDlgItem(IDC_SWSPTM_EDIT)->EnableWindow(FALSE);}
      else{m_SWEnableDisable="Timing\nDisabled";
    GetDlgItem(IDC_SWSTTM_EDIT)->EnableWindow(TRUE);
    GetDlgItem(IDC_SWSPTM_EDIT)->EnableWindow(TRUE);}

      if(m_EnableShutdown==TRUE)
      { m_ESD="Shutdown\nEnabled";
      GetDlgItem(IDC_SHUTDOWNTIME)->EnableWindow(FALSE);}
      else {m_ESD="Shutdown\nDisabled";
              GetDlgItem(IDC_SHUTDOWNTIME)->EnableWindow(TRUE);}
    UpdateData(FALSE);
              SaveSettings();
}




void CDeXDlg::TimmerEvents()
{
        UpdateData(TRUE);
if(m_BBEnableTiming==TRUE)
        { if(strcmp(m_sTime,m_BBStartTime)==0) m_pin2=1;
  if(strcmp(m_sTime,m_BBStopTime)==0) m_pin2=0;}

if(m_SWEnableTiming==TRUE)
        {if(strcmp(m_sTime,m_SWStartTime)==0) m_pin3=1;
  if(strcmp(m_sTime,m_SWStopTime)==0) m_pin3=0;}

if(m_GTEnableTiming==TRUE)
        {if(strcmp(m_sTime,m_GTStartTime)==0) m_pin4=1;
        if(strcmp(m_sTime,m_GTStopTime)==0) m_pin4=0;}

if(m_BBEnableTiming==TRUE)
        {if(strcmp(m_sTime,m_SSStartTime)==0) m_pin5=1;
        if(strcmp(m_sTime,m_SSStopTime)==0) m_pin5=0;}

if(m_EnableShutdown==TRUE)
        {if(strcmp(m_sTime,m_ShutDownTime)==0)
        ShutDown();}

UpdateData(FALSE);
ChangePin();
}
```

```
void CDeXDlg::UpdatePins()
{
 int reg;

     reg=Inp32(DATA);

     if((reg & 0x01)==0) m_pin2=0;    else m_pin2=1;
     if((reg & 0x02)==0) m_pin3=0;    else m_pin3=1;
     if((reg & 0x04)==0) m_pin4=0;    else m_pin4=1;
     if((reg & 0x08)==0) m_pin5=0;    else m_pin5=1;

      UpdateData(FALSE);
}

void CDeXDlg::ChangePin()
{
int data_register, new_register;

UpdateData(TRUE);
data_register=Inp32( DATA );
new_register=0;
if( m_pin2==TRUE ) new_register |= 0x01;
if( m_pin3==TRUE ) new_register |= 0x02;
if( m_pin4==TRUE ) new_register |= 0x04;
if( m_pin5==TRUE ) new_register |= 0x08;

Out32(DATA, new_register);

          if(m_pin2==1) m_BBOnOff="On"; else m_BBOnOff="Off";
   if(m_pin3==1) m_SWOnOff="On"; else m_SWOnOff="Off";
   if(m_pin4==1) m_GTOnOff="On"; else m_GTOnOff="Off";
   if(m_pin5==1)  {

                                          }
                     else
                                          {};

   UpdateData(FALSE);
SaveSettings();

}

void CDeXDlg::LoadSettings()
{
        fstream fp;
        fp.open("DeCON",ios::in);
        fp>>m_BBEnableTiming>>m_SWEnableTiming>>m_SSEnableTiming>>m_GTEnableTi
ming;
   UpdateData(FALSE);
        fp.close();


        fp.open("DeCON1",ios::in);
        fp>>m_pin2>>m_pin3>>m_pin4>>m_pin5;
   UpdateData(FALSE);
        fp.close();
    EnableTiming();
  m_pin9=1;
   ChangePin();
}

void CDeXDlg::SaveSettings()
{
     fstream fp;
                fp.open("DeCON",ios::out);
        fp<<m_BBEnableTiming<<setw(5)<<m_SWEnableTiming<<setw(5)<<m_SSEnableTimin
g<<setw(5)<<m_GTEnableTiming;
                fp.close();

    fp.open("DeCON1",ios::out);
        fp<<m_pin2<<setw(5)<<m_pin3<<setw(5)<<m_pin4<<setw(5)<<m_pin5;
        fp.close();

                UpdateData(TRUE);
}

void CDeXDlg::OnPin2()
{
        // TODO: Add your control notification handler code here
        ChangePin();

}

void CDeXDlg::OnPin3()
{
        // TODO: Add your control notification handler code here
        ChangePin();
}

void CDeXDlg::OnPin4()
{
        // TODO: Add your control notification handler code here
        ChangePin();
}

void CDeXDlg::OnPin5()
{
        // TODO: Add your control notification handler code here

   m_pin5=FALSE;

   m_SSOnOff="Off";
        UpdateData(FALSE);

        ChangePin();
}

void CDeXDlg::ShutDown()
{



  WinExec("shutdown.exe",SW_SHOW);
}

void CDeXDlg::OnEnableshutdown()
{
        // TODO: Add your control notification handler code here
        EnableTiming();
}

void CDeXDlg::StatusPins()
{
  int status_reg;
  status_reg=Inp32(STATUS);

  if((status_reg & 0x40)==0) m_BBApplianceStatus ="Not working";
  else  m_BBApplianceStatus ="Working";

  if((status_reg & 0x80)==0) m_SWApplianceStatus ="Not working";
        else m_SWApplianceStatus ="Working";

  if((status_reg & 0x20)==0) m_GTApplianceStatus ="Not working";
        else m_GTApplianceStatus ="Working";

  if((status_reg & 0x10)==0) m_SSApplianceStatus  ="Not working";
        else m_SSApplianceStatus ="Working";

}
```

```
BOOL CDeXDlg::OnHelpInfo(HELPINFO* pHelpInfo)
{
	// TODO: Add your message handler code here and/or call default

	return CDialog::OnHelpInfo(pHelpInfo);
}

void CDeXDlg::OnPin6()
{
	// TODO: Add your control notification handler code here
	m_pin5=TRUE;
  m_pin7=TRUE;
  m_pin8=TRUE;

  m_SSOnOff="1";
	UpdateData(FALSE);

	ChangePin();
}

void CDeXDlg::OnPin7()
{
	// TODO: Add your control notification handler code here

  m_pin5=TRUE;
  m_pin6=TRUE;
  m_pin8=TRUE;
  m_SSOnOff="2";
  UpdateData(FALSE);
	ChangePin();

}

void CDeXDlg::OnPin8()
{
	// TODO: Add your control notification handler code here

  m_pin5=TRUE;
  m_pin6=TRUE;
  m_pin7=TRUE;
  m_SSOnOff="3";
  UpdateData(FALSE);
	ChangePin();
}
```

LIST OF FIGURES

LIST OF TABLES

REFERENCES

[1]. C++ Programming Language, available at: www.mcp.com

[2]. Norris, Mark. "Microcomputer". Microsoft Encarta (c) 2006

[3]. C++ Programming Language, available at: www.mcp.com

[4]. C++ Programming Language, available at: www.mcp.com

[5]. C++ Programming Language, available at: www.mcp.com

[6]. Harsha, Perla: Programming Language, available at: www.electrosoft.com

[7]. Harsha, Perla: Programming Language, available at: www.electrosoft.com